
Photon Documentation

Release 0.4

Frieder Griesshammer

October 14, 2015

1	Photon Intro	3
1.1	Examples	3
2	Installation	5
3	Structure	7
3.1	The core	7
3.2	Tools	14
3.3	Utility	21
4	Info	29
	Python Module Index	31



Welcome to the Photon Documentation.

Photon Intro

It could be best described as a **shell backend as python module**

Contributions are highly welcome ¹, also feel free to use the [issue tracker](#) if you encounter any problems.

Repository github.com/spookey/photon

Documentation photon.readthedocs.org

Package pypi.python.org/pypi/photon_core

1.1 Examples

The `/examples` directory contains some basic receipts on how to use Photon in your scripts.

Photon helps at [Freifunk MWU](#) to solve some tasks:

- See our [collection of backend-scripts](#) for some scripts using photon, running in production.
- To automatically compile gluon firmware for routers, we wrote the [gluon builder](#).

¹ Teach me how to write good code, help me to improve.

Installation

Photon is available as package on pypi, it is called `photon_core` ¹.

You can install/update the package via `pip3` ²:

```
pip3 install photon_core
```

```
pip3 install -U photon_core
```

Bleeding-Edge

Development is still at an very early stage, expect anything to change completely in near future.

As long we still have a leading zero in the version (see *info* file) use *pip3* with the `--pre` switch:

```
pip3 install -U photon_core --pre
```

Versions

Tags in the git repository will be released as a new pypi package version. Versions of a pypi package has always it's git tag. And vice versa.

Not every version increase will be tagged/released. I will only do so if I feel the urge to do so.

¹ because photon itself was already taken :/

² Photon is written in python3 ~ be careful with `easy_install`

Structure

Photon aims to be modular and can be divided into *The core*, it's *Utility* and some *Tools*, provided through *Photon* itself.

If you just want to use Photon in your Scripts as a normal User you may especially be interested in the parts *Photon* and *Tools*.

3.1 The core

All three modules depend on the *Utility*:

See also:

Files, Locations, Structures, System

Settings and *Meta* could be used independently or both together.

Bundling *Settings* and *Meta* together plus adding the *Tools*, *Photon* provides a interface to use in your scripts.

See also:

Git Tool, Mail Tool, Ping Tool, Signal Tool

3.1.1 Settings

class settings.**Settings** (*defaults, config='config.yaml', verbose=True*)

Settings is a class which provides access to compiled settings loaded from YAML-files.

The YAML-files will be read with specific loaders which enables certain logic within the configuration. It is possible to:

- Insert references to existing fields via anchors and `!str_join` or `!loc_join`
- Insert keywords like **hostname** or **timestamp** using `!str_join`
- Combine path-segments using `!loc_join`
- Insert keywords like **home_dir** or **conf_dir** using `!loc_join`

It is also possible to import or merge further content.

Parameters

- **defaults** – The initial configuration to load. Will be located using `util.locations.search_location()`

- The common way is to use a short-filename to locate it next to the script using Photon.
 - Can also be a full path.
 - Can also passed directly as a dict
 - Bring your own defaults! Tears down (using `util.system.shell_notify()` with `state` set to `True`) whole application if not found or none passed.
 - **config** – Where to store the loaded output from the *defaults*. Will be located using `util.locations.search_location()`
 - File must already exist, will be created in ‘conf_dir’ from `util.locations.get_locations()` otherwise
 - * Therefore use a short name (or full path) if one should be created
-

Note: The last loaded file wins

- The config is intended to provide a editable file for the end-user
 - If a value differs from the original values in *defaults*, the value in *config* wins
 - * Other values which not exist in *config* will be set from *defaults*
 - * If a value in *config* contains a loader call which expresses the same as the value in *defaults* it will be skipped.
 - Be careful using **timestamp** s in a config. The timestamp of the first launch will always be used.
 - Simply delete all lines within the config to completely reset it to the defaults
-
- Can be skipped by explicitly setting it to `None`
- **verbose** – Sets the *verbose* flag for the underlying *Utility* functions

See also:

`util.structures.yaml_str_join()` and `util.structures.yaml_loc_join()` as well as the *Example Settings File*

get

Returns Current settings

load (*skey*, *sdesc*, *sdict*=None, *loaders*=None, *merge*=False, *writeback*=False)

Loads a dictionary into current settings

Parameters

- **skey** – Type of data to load. Is be used to reference the data in the files sections within settings
- **sdesc** – Either filename of yaml-file to load or further description of imported data when *sdict* is used
- **sdict** (*dict*) – Directly pass data as dictionary instead of loading it from a yaml-file. Make sure to set *skey* and *sdesc* accordingly
- **loaders** (*list*) – Append custom loaders to the YAML-loader.
- **merge** – Merge received data into current settings or place it under *skey* within meta

- **writeback** – Write back loaded (and merged/imported) result back to the original file.
This is used to generate the summary files

Returns The loaded (or directly passed) content

See also:

`util.structures.yaml_str_join()` and `util.structures.yaml_loc_join()`

Example Settings File

defaults.sample.yaml

```

1  # The syntax of the settings files is YAML:
2
3  01_syntax:
4      dictionary: 'value is a string'
5      dictionary_with_list: ['value', 'is', 'a', 'list']
6      dictionary_with_list2:
7          - this
8          - is
9          - another
10         - list
11
12
13  # ----
14
15  # YAML supports backreferences by anchors.
16
17  # First yo have define a dictionary value as anchor:
18  02_anchor:
19      prefix: &MY_PRFX 'Photon is a software that '
20
21  # Then use them together with !str_join:
22  poll:
23      yay: !str_join [*MY_PRFX, 'really helps me']
24      nay: !str_join [*MY_PRFX, 'sucks']
25
26
27  # This turns into:
28  #     yay: Photon is a software that really helps me
29  #     nay: Photon is a software that sucks
30  # (The anchor ('&'-sign) must appear before the Reference ('*-sign) in the YAML-file.
31  # (Note the whitespace.)
32
33
34  # ----
35
36  # !str_join can listen to the keywords - 'hostname' & 'timestamp':
37
38  03_keywords:
39      message:
40          - !str_join ['my machine "', 'hostname', '" is the best']
41          - !str_join ['yours, herbert. date: ', 'timestamp']
42
43
44  # This turns into:
45  #     message:

```

```
46 # - my machine "blechschachtel" is the best
47 # - 'yours, herbert. date: YYYY.MM.DD-HH.MM.SS'
48 # (with current date expanded)
49
50
51 # ----
52
53 # Use !loc_join to combine files and paths:
54
55 04_locations:
56     simple_file: !loc_join ['/', 'usr', 'local', 'bin', 'myscript.sh']
57     same_simple_file: !loc_join ['/usr/local/bin', 'myscript.sh']
58 # This turns into:
59 #     simple_file: /usr/local/bin/myscript.sh
60 #     same_simple_file: /usr/local/bin/myscript.sh
61
62 # But be careful with leading '/'-signs:
63     not_the_simple_file: !loc_join ['/usr/local', '/bin', 'myscript.sh']
64 # This turns into not what we wanted:
65 #     not_the_simple_file: /bin/myscript.sh
66
67
68 # It can also listen to keywords:
69     in_the_home_dir: !loc_join ['home_dir', 'my_directory']
70 #     in_the_home_dir: /home/herbert/my_directory
71
72
73 # ----
74
75 # Combine them altogether:
76
77 05_combined:
78     name: &MY_ASS my_awesome_server_software
79
80     main: &OH_MY !loc_join ['home_dir', *MY_ASS, 'main']
81
82     main_run: !loc_join [*OH_MY, 'run.py']
83
84     backup_dir: !loc_join ['data_dir', *MY_ASS, !str_join ['backup-', 'timestamp']]
85
86     git-remote: !str_join
87     - 'https://github.com/user404/'
88     - *MY_ASS
89     - .git
90
91 # This turns into:
92 #     name: my_awesome_server_software
93 #     main: /home/herbert/my_awesome_server_software/main
94 #     main_run: /home/herbert/my_awesome_server_software/main/run.py
95 #     backup_dir: /home/herbert/.local/share/photon/my_awesome_server_software/backup-YYYY.MM.DD-HH.MM.SS
96 #     git-remote: https://github.com/user404/my_awesome_server_software.git
```

See also:

The [wikipedia page on YAML](#) for some syntax reference.

See also:

- `!loc_join: util.structures.yaml_loc_join()`

(get locations by keyword and join paths)

- `!str_join: util.structures.yaml_str_join()`

(get variables by keyword and join strings)

See also:

Example Settings File, Mail Tool Example, Ping Tool Example

3.1.2 Meta

class `meta.Meta` (`meta='meta.json', verbose=True`)

Meta is a class which bounds to an actual json-file on disk. It provides a logger storing the entries in that json-file.

It is also possible to import contents. By staging out to a different directory meta-files are left behind for further debugging or to see what was going on.

Parameters

- **meta** – Initial, clean meta file to use. See `stage()` for more
- **verbose** – Sets the `verbose` flag for the underlying *Utility* functions

load (`mkey, mdesc, mdict=None, merge=False`)

Loads a dictionary into current meta

Parameters

- **mkey** – Type of data to load. Is be used to reference the data from the ‘header’ within meta
- **mdesc** – Either filename of json-file to load or further description of imported data when `mdict` is used
- **mdict** (*dict*) – Directly pass data as dictionary instead of loading it from a json-file. Make sure to set `mkey` and `mdesc` accordingly
- **merge** – Merge received data into current meta or place it under ‘import’ within meta

Returns The loaded (or directly passed) content

log

Parameters **elem** – Add a new log entry to the meta.

- Can be anything.
- The `log` is a dictionary with keys generated from the output of `util.system.get_timestamp()` and `elem` as value

Returns Current meta

stage (`name, clean=False`)

Switch stage

Parameters

- **name** – Filename of new meta file. Will be located using `util.locations.search_location()`
- File must not already exist, will be created in ‘data_dir’ from `util.locations.get_locations()`
- Can also be a full path to place it anywhere desired

- **clean** – What to do with preexisting meta files?
 - `False`: Merge current meta with preexisting one
 - `True`: Replace preexisting meta with current one

3.1.3 Photon

`class photon.Photon (defaults, config='config.yaml', meta='meta.json', verbose=True)`

Photon uses *The core* and some functions from *Utility* in its `m()`-method.

The `m()`-method itself is used in each tool to interact with photon to:

- Launch shell commands, and receive the results
- Add messages to the *meta*-file
- Show the messages if necessary
- Tear down application completely in case of any serious problems

Further, Photon provides direct handlers for *settings.Settings* and *meta.Meta* and a handler for each tool from *Tools* by it's methods.

Parameters

- **defaults** – Pass *defaults* down to *settings.Settings*
- **config** – Pass *config* down to *settings.Settings*
- **meta** – Pass *meta* down to *meta.Meta*
- **verbose** – Sets the global *verbose* flag. Passes it down to the underlying *Utility* functions and *The core*

Variables

- *settings* – The settings handler initialized with *defaults* and *config*
- *meta* – The meta handler initialized with *meta*

At startup the loaded *settings* are imported into *meta*

`git_handler (*args, **kwargs)`

Returns A new git handler

See also:

Git Tool

`m (msg, state=False, more=None, cmdd=None, critical=True, verbose=None)`

Mysterious mega method managing multiple meshed modules magically

Note: If this function is used, the code contains facepalms: `m (`

- It is possible to just show a message, or to run a command with message.
- But it is not possible to run a command without a message, use the *verbose*-flag to hide your debug message.

Parameters

- **msg** – Add a message. Shown depending on *verbose* (see below)

- **state** – Pass *state* down to `util.system.shell_notify()`
- **more** – Pass *more* down to `util.system.shell_notify()`
- **cmd** (*dict*) – If given, `util.system.shell_run()` is launched with it's values
- **critical** – If set to `True`: Tears down (using `util.system.shell_notify()` with *state* set to `True`) whole application on failure of *cmd* contents.
 - Similar to `util.system.shell_run()` *critical*-flag
- **verbose** – Overrides parent's class *verbose*-flag.
 - If left to `None`, the verbose value Photon was started with is used
 - Messages are shown/hidden if explicitly set to `True/False`

Returns

A dictionary specified the following:

- **'more'**: *more* if it is not a dictionary otherwise it gets merged in if *more* is specified
- The output of `util.system.shell_run()` gets merged in if *cmd* is specified
- **'failed'**: `True` if command failed

`util.system.shell_notify()` is used with this dictionary to pipe it's output into `meta.Meta.log()` before returning.

mail_handler (*punchline=None, add_meta=False, add_settings=True, *args, **kwargs*)

Parameters

- **punchline** – Adds a punchline before further text
- **add_meta** – Appends current meta to the mail
- **add_settings** – Appends current settings to the mail

Returns A new mail handler

See also:

Mail Tool

ping_handler (**args, **kwargs*)

Returns A new ping handler

See also:

Ping Tool

s2m

Imports settings to meta

signal_handler (**args, **kwargs*)

Returns A new signal handler

See also:

Signal Tool

template_handler (**args, **kwargs*)

Returns A new template handler

See also:

Template Tool

`photon.check_m(pm)`

Shared helper function for all *Tools* to check if the passed m-function is indeed `photon.Photon.m()`

Params `pm` Suspected m-function

Returns Now to be proven correct m-function, tears down whole application otherwise.

3.2 Tools

This are the tools for the user using Photon. You should not directly use them, instead they will get provided to you by *Photon*.

See also:

Settings, Meta, Photon

Some functionality here is bought from the *Utility*:

See also:

Files, Locations, Structures, System

3.2.1 Git Tool

`class tools.git.Git(m, local, remote_url=None, mbranch=None)`

The git tool helps to deal with git repositories.

Parameters

- **local** – The local folder of the repository
 - If `None` given (default), it will be ignored if there is already a git repo at *local*
 - If no git repo is found at *local*, a new one gets cloned from *remote_url*
- **remote_url** – The remote URL of the repository
 - Tears down (using `util.system.shell_notify()` with *state* set to `True`) whole application if *remote_url* is set to `None` but a new clone is necessary
- **mbranch** – The repository's main branch. Is set to *master* when left to `None`

`_checkout(treeish)`

Helper function to checkout something

Parameters `treeish` – String for 'tag', 'branch', or remote tracking '-B *branch*'

`_get_branch(remotes=False)`

Helper function to determine current branch

Parameters `remotes` – List the remote-tracking branches

`_get_remote(cached=True)`

Helper function to determine remote

Parameters `cached` – Use cached values or query remotes

`_log(num=None, format=None)`

Helper function to receive git log

Parameters

- **num** – Number of entries
- **format** – Use formatted output with specified format string

`_pull()`

Helper function to pull from remote

`branch`

Parameters **branch** – Checks out specified branch (tracking if it exists on remote). If set to `None`, ‘master’ will be checked out

Returns The current branch (This could also be ‘master (Detatched-Head)’ - Be warned)

`cleanup`

Commits all local changes (if any) into a working branch, merges it with ‘master’.

Checks out your old branch afterwards.

Tears down (using `util.system.shell_notify()` with `state` set to `True`) whole application if conflicts are discovered

`commit`

Parameters **tag** – Checks out specified commit. If set to `None` the latest commit will be checked out

Returns A list of all commits, descending

`local`

Returns The local folder of the repository

`log`

Returns The last 10 commit entries as dictionary

- ‘commit’: The commit-ID
- ‘message’: First line of the commit message

`publish`

Runs `cleanup()` first, then pushes the changes to the `remote`.

`remote`

Returns Current remote

`remote_url`

Returns The remote URL of the repository

`short_commit`

Returns A list of all commits, descending

See also:

`commit`

`status`

Returns Current repository status as dictionary:

- ‘clean’: `True` if there are no changes `False` otherwise

- ‘untracked’: A list of untracked files (if any and not ‘clean’)
- ‘modified’: A list of modified files (if any and not ‘clean’)
- ‘deleted’: A list of deleted files (if any and not ‘clean’)
- ‘conflicting’: A list of conflicting files (if any and not ‘clean’)

tag

Parameters **tag** – Checks out specified tag. If set to `None` the latest tag will be checked out

Returns A list of all tags, sorted as version numbers, ascending

3.2.2 Mail Tool

class `tools.mail.Mail` (*m*, *to*, *sender*, *subject=None*, *cc=None*, *bcc=None*)

The Mail tool helps to send out mails.

Parameters

- **to** – Where to send the mail (`‘user@example.com’`)
- **sender** – Yourself (`‘me@example.com’`)
 - set a reverse DNS entry for `example.com` so your mail does not get caught up in spamfilters.
- **subject** – The subject line
- **cc** – One or a list of CCs
- **bcc** – One or a list of BCCs

send

Returns

A dictionary with the following:

- ‘sender’: The *sender*
- ‘recipients’: All recipients, compiled from *to*, *cc* and *bcc*
- ‘result’: The `smtplib.SMTP.sendmail()`-result
- ‘exception’: The exception message (if any)

Note: You need to have a postfix/sendmail running and listening on localhost.

text

Parameters **text** – Add some more text

Returns All text & headers as raw mail source

Mail Tool Example

`mail.sample.yaml`

```

1 mail:
2     recipient: you@example.com
3     sender: me@example.com
4     subject: 'Fire!'
5     punchline: 'Dear Sir or Madam, I am writing to inform you about a fire in the building ...'

```

mail.sample.py

```

1 from photon import Photon
2
3 photon = Photon('mail.sample.yaml')
4
5 settings = photon.settings.get['mail']
6
7 mail = photon.mail_handler(
8     to=settings['recipient'],
9     sender=settings['sender'],
10    subject=settings['subject'],
11    punchline=settings['punchline'],
12    add_meta=True
13 )
14
15 ###
16 # Shows the message source so far
17 print(mail.text)
18
19 ###
20 # Add some more text (do this as often as you like):
21 mail.text = '''
22 Dear Sir or Madam,
23 bla bla
24
25 No, that's too formal..
26 '''
27
28 ###
29 # Guess what happens here:
30 mail.send

```

See also:

Example Settings File, Mail Tool Example, Ping Tool Example

3.2.3 Ping Tool

class tools.ping.**Ping**(m, six=False, net_if=None, num=5, max_pool_size=None)

The Ping tool helps to send pings, returning detailed results each probe, and calculates a summary of all probes.

Parameters

- **six** – Either use ping or ping6
- **net_if** – Specify network interface to send pings from
- **num** – How many pings to send each probe

- **max_pool_size** – Hosts passed to `probe()` in form of a list, will be processed in parallel. Specify the maximum size of the thread pool workers here. If skipped, the number of current CPUs is used

probe

Parameters **hosts** – One or a list of hosts (URLs, IP-addresses) to send pings to

- If you need to check multiple hosts, it is best to pass them together as a list.
- This will probe all hosts in parallel, with `max_pool_size` workers.

Returns A dictionary with all hosts probed as keys specified as following:

- `'up'`: True or False depending if ping was successful
- `'loss'`: The packet loss as list (if `'up'`)
- `'ms'`: A list of times each packet sent (if `'up'`)
- `'rtt'`: A dictionary with the fields *avg*, *min*, *max* & *stddev* (if `'up'`)

status

Returns A dictionary with the following:

- `'num'`: Total number of hosts already probed
- `'up'`: Number of hosts up
- `'down'`: Number of hosts down
- `'ratio'`: Ratio between `'up'/'down'` as float

Ratio:

- 100% up == *1.0*
- 10% up == *0.1*
- 0% up == *0.0*

Ping Tool Example

ping.sample.yaml

```
1 hosts:
2   addresses:
3     - '127.0.0.1'
4     - '127.0.0.2'
5     - '127.0.0.3'
6   urls:
7     - exampla.com
8     - example.com
9     - exampli.com
10    - exemplo.com
11    - examplu.com
```

ping.sample.py

```

1  from pprint import pprint
2  from photon import Photon
3
4  photon = Photon('ping.sample.yaml')
5
6  hosts = photon.settings.get['hosts']
7
8  ping = photon.ping_handler()
9
10 ###
11 # Let's start off with localhost to demonstrate the handling of the probe-function:
12
13 pprint(hosts)
14
15 a = hosts['addresses'][0]
16 ping.probe = a
17
18 if ping.probe[a]['up']:
19     print('%s is reachable - %s ms rtt in average' %(a, ping.probe[a]['rtt']['avg']))
20 else:
21     print('%s could not be reached!' %(a))
22
23 pprint(ping.probe)
24
25 print('-' * 8)
26
27
28 ###
29 # You can also pass a complete list to probe. This will be faster, because the list is processed in p
30 # The status per host will be overwritten with new information if it encounters the same host again:
31
32 ping.probe = hosts['addresses']
33 pprint(ping.probe)
34
35 print('These are the statistics so far:')
36 pprint(ping.status)
37
38 print('-' * 8)
39
40
41 ###
42 # Another round of pings to demonstrate the handling of the status-function:
43
44 ping.probe = hosts['urls']
45
46 if ping.status['ratio'] <= 0.75:
47     print('more than three quarters of all addresses are not reachable!!!')
48
49 print('The statistics have changed now:')
50 pprint(ping.status)

```

See also:

Example Settings File, Mail Tool Example, Ping Tool Example

3.2.4 Signal Tool

`class tools.signal.Signal(m, pid, sudo=True, cmd_if_no_pid=None)`

The Signal tool can send signals to processes via `kill`, returning the results.

Parameters

- **pid** – Either the full path to the pidfile (e.g. `/var/run/proc.pid`) or the pid as number
- **sudo** – Prepend `sudo` before command. (Make sure to be root yourself if set to `False` or expect errors. Further for unattended operation add the user to `sudoers` file.)

`__Signal__signal(sig, verbose=None)`

Helper class preventing code duplication..

Parameters

- **sig** – Signal to use (e.g. “HUP”, “ALRM”)
- **verbose** – Overwrite `photon.Photon.m()`’s `verbose`

Returns `photon.Photon.m()`’s result of killing `pid` with specified `pid`

alarm

Returns `photon.Photon.m()`’s result of killing `pid` using SIGALRM

hup

Returns `photon.Photon.m()`’s result of killing `pid` using SIGHUP

int

Returns `photon.Photon.m()`’s result of killing `pid` using SIGINT with visible shell warning

kill

Returns `photon.Photon.m()`’s result of killing `pid` using SIGKILL with visible shell warning

quit

Returns `photon.Photon.m()`’s result of killing `pid` using SIGQUIT with visible shell warning

stop

Returns `photon.Photon.m()`’s result of killing `pid` using SIGSTOP with visible shell warning

usr1

Returns `photon.Photon.m()`’s result of killing `pid` using SIGUSR1

usr2

Returns `photon.Photon.m()`’s result of killing `pid` using SIGUSR2

3.2.5 Template Tool

`class tools.template.Template(m, template, fields=None)`

The Template tool helps to process on strings.

Parameters

- **template** – The initial template to start with.
 - If it's value is recognized by `util.locations.search_location()` (a.k.a is a filename) the file contents will be loaded as template.

Note: If the file is not found, you will be doing string processing on the filename instead of the contents!

- **fields** – Initially set up fields. Can be done later, using `sub()`

The templating-language itself are normal [Template strings](#), see there for syntax.

raw

Returns The raw template

sub

Parameters **fields** – Set fields to substitute

Returns Substituted Template with given fields. If no fields were set up beforehand, `raw()` is used.

write (*filename*, *append=True*, *backup=True*)

Parameters

- **filename** – File to write into
- **append** – Either append to existing content (if not already included) or completely replace *filename*
- **backup** – Create a backup of *filename* before writing. Only applies when *append* is set

3.3 Utility

This is the toolbox used by *The core*:

See also:

Settings, Meta, Photon

As well as used by the *Tools*:

See also:

Git Tool, Mail Tool, Ping Tool, Signal Tool

Note: If you have no explicit reason, do **not** use the functions here directly.

- Always try to work through `photon.Photon` and it's handlers.
- If you discover you are repeatedly calling backend functions

consider adding a tool for that job!

3.3.1 Files

`util.files.read_file(filename)`

Reads files

Parameters `filename` – The full path of the file to read

Returns The content of the file as string (if `filename` exists)

Note: If `filename`'s content is empty, `None` will also returned.

To check if a file really exists use `util.locations.search_location()`

`util.files.read_json(filename)`

Reads json files

Parameters `filename` – The full path to the json file

Returns Loaded json content as represented data structure

`util.files.read_yaml(filename, add_constructor=None)`

Reads YAML files

Parameters

- **filename** – The full path to the YAML file
- **add_constructor** – A list of yaml constructors (loaders)

Returns Loaded YAML content as represented data structure

See also:

`util.structures.yaml_str_join()`, `util.structures.yaml_loc_join()`

`util.files.write_file(filename, content)`

Writes files

Parameters

- **filename** – The full path of the file to write (enclosing folder must already exist)
- **content** – The content to write

Returns The size of the data written

`util.files.write_json(filename, content)`

Writes json files

Parameters

- **filename** – The full path to the json file
- **content** – The content to dump

Returns The size written

`util.files.write_yaml(filename, content)`

Writes YAML files

Parameters

- **filename** – The full path to the YAML file
- **content** – The content to dump

Returns The size written

3.3.2 Locations

`util.locations.backup_location(src, loc=None)`

Writes Backups of locations

Parameters

- **src** – The source file/folder to backup
 - **loc** – The target folder to backup into
- The backup will be called `src + util.system.get_timestamp()`. * If `loc` left to none, the backup gets written in the same folder like `src` resides in
- Otherwise the specified path will be used.

`util.locations.change_location(src, tgt, move=False, verbose=True)`

Copies/moves/deletes locations

Parameters

- **src** – Source location where to copy from
- **tgt** – Target location where to copy to
 - To backup `src`, set `tgt` explicitly to `True`. `tgt` will be set to `src + '_backup_' + util.system.get_timestamp()` then
- **move** – Deletes original location after copy (a.k.a. move)
 - To delete `src`, set `tgt` explicitly to `False` and `move` to `True` (be careful!!!)
- **verbose** – Show warnings

`util.locations.get_locations()`

Compiles default locations

Returns A dictionary with folders as values:

- **'home_dir'**: Your home-directory (~)
- **'call_dir'**: Where you called the first Python script from. (`argv[0]`)
- **'conf_dir'**: The `XdG_CONFIG_HOME`-directory + photon (`~/.config/photon`)
- **'data_dir'**: The `XdG_DATA_HOME`-directory + photon (`~/.local/share/photon`)

Note:

- Both `search_location()` and `make_locations()` have the argument `locations`.
 - If `locations` is set to `None` (by default), it will be filled with the output of `get_locations()`.
-

`util.locations.make_locations(locations=None, verbose=True)`

Creates folders

Parameters

- **locations** – A list of folders to create (can be a dictionary, see note below)
- **verbose** – Warn if any folders were created

Note:

- If *locations* is not a list, but a dictionary, all values in the dictionary will be used (as specified in `util.structures.to_list()`)
 - If *locations* is set to `None` (by default), it will be filled with the output of `get_locations()`.
-

`util.locations.search_location(loc, locations=None, critical=False, create_in=None, verbose=True)`

Locates files with a twist:

- Check the existence of a file using the full path in *loc*
- Search for the filename *loc* in *locations*
- Create it's enclosing folders if the file does not exist. use *create_in*

Parameters

- **loc** – Filename to search
- **locations** – A list of possible locations to search within (can be a dictionary, see note below)
- **critical** – Tears down (using `util.system.shell_notify()` with *state* set to `True`) whole application if file was not found
- **create_in** – If *loc* was not found, the folder *create_in* is created. If *locations* is a dictionary, *create_in* can also specify a key of *locations*. The value will be used then.
- **verbose** – Pass verbose flag to `make_locations()`

Returns The full path of *loc* in matched location

Note:

- If *locations* is not a list, but a dictionary, all values in the dictionary will be used (as specified in `util.structures.to_list()`)
 - If *locations* is set to `None` (by default), it will be filled with the output of `get_locations()`.
-

3.3.3 Structures

`util.structures.dict_merge(o, v)`

Recursively climbs through dictionaries and merges them together.

Parameters

- **o** – The first dictionary
- **v** – The second dictionary

Returns A dictionary (who would have guessed?)

Note: Make sure *o* & *v* are indeed dictionaries, bad things will happen otherwise!

`util.structures.to_list(i, use_keys=False)`

Converts items to a list.

Parameters

- **i** – Item to convert

- If *i* is `None`, the result is an empty list
- If *i* is `'string'`, the result won't be `['s', 't', 'r', ...]` rather more like `['string']`
- If *i* is a nested dictionary, the result will be a flattened list.
- **use_keys** – If *i* is a dictionary, use the keys instead of values

Returns All items in *i* as list

`util.structures.yaml_loc_join(l, n)`

YAML loader to join paths

The keywords come directly from `util.locations.get_locations()`. See there!

Returns A *path separator* (`/`) joined string with keywords extended. Used in `settings.Settings.load()`

See also:

The YAML files mentioned in [Example Settings File](#), [Mail Tool Example](#), [Ping Tool Example](#)

`util.structures.yaml_str_join(l, n)`

YAML loader to join strings

The keywords are as following:

- **hostname**: Your hostname (from `util.system.get_hostname()`)
- **timestamp**: Current timestamp (from `util.system.get_timestamp()`)

Returns A *non character* joined string with keywords extended. Used in `settings.Settings.load()`

Note: Be careful with timestamps when using a *config* in [Settings](#).

See also:

The YAML files mentioned in [Example Settings File](#), [Mail Tool Example](#), [Ping Tool Example](#)

3.3.4 System

`util.system.get_hostname()`

Determines the current hostname by probing `uname -n`. Falls back to `hostname` in case of problems.

Tears down (using `util.system.shell_notify()` with *state* set to `True`) whole application if both failed (usually they don't but consider this if you are debugging weird problems..)

Returns The hostname as string. Domain parts will be split off

`util.system.get_timestamp(time=True, precice=False)`

What time is it?

Parameters

- **time** – Append `-%H.-%M.-%S` to the final string.
- **precice** – Append `-%f` to the final string. Is only recognized when *time* is set to `True`

Returns A timestamp string of now in the format `%Y.-%m.-%d-%H.-%M.-%S-%f`

See also:

strftime.org is awesome!

`util.system.shell_notify(msg, state=False, more=None, exitcode=None, verbose=True)`

A pretty long wrapper for a `print()` function. But this `print()` is the only one in Photon.

Note: This method is just a helper method within photon. If you need this functionality use `photon.Photon.m()` instead

Parameters

- **msg** – The message to show
- **state** – The message will be prefixed with `[state]`
 - If `False` (default): Prefixed with `~`
 - If `None`: Prefixed with `[WARNING]`
 - If `True`: Prefixed with `[FATAL]` and the `exitcode` will be set (see below)
- **more** – Something to add to the message (see below)
 - Anything you have. Just for further information.
 - Will be displayed after the message, pretty printed using `pprint.pformat()`
- **exitcode** – Tears down (using `util.system.shell_notify()` with `state` set to `True`) whole application with given code
- **verbose** – Show message or not (see below)
 - If set to `False`, you can use `shell_notify()` for the dictionary it returns.
 - Will be overruled if `exitcode` is set.

Returns A dictionary containing untouched `msg`, `more` and `verbose`

`util.system.shell_run(cmd, cin=None, cwd=None, timeout=10, critical=True, verbose=True)`

Runs a shell command within a controlled environment.

Note: This method is just a helper method within photon. If you need this functionality use `photon.Photon.m()` instead

Parameters

- **cmd** – The command to run
 - A string one would type into a console like `git push -u origin master`.
 - Will be split using `shlex.split()`.
 - It is possible to use a list here, but then no splitting is done.
- **cin** – Add something to stdin of `cmd`
- **cwd** – Run `cmd` inside specified current working directory
- **timeout** – Catch infinite loops (e.g. ping). Exit after `timeout` seconds
- **critical** – If set to `True`: Tears down (using `util.system.shell_notify()` with `state` set to `True`) whole application on failure of `cmd`
- **verbose** – Show messages and warnings

Returns

A dictionary containing the results from running *cmd* with the following:

- 'command': *cmd*
- 'stdin': *cin* (If data was set in *cin*)
- 'cwd': *cwd* (If *cwd* was set)
- 'exception': exception message (If an exception was thrown)
- 'timeout': *timeout* (If a timeout exception was thrown)
- 'stdout': List from stdout (If any)
- 'stderr': List from stderr (If any)
- 'returncode': The returncode (If not any exception)
- 'out': The most urgent message as joined string. ('exception' > 'stderr' > 'stdout')

I am lost:

- genindex
- modindex
- search

Info

The *info* file

The *info* file is not vital to Photon, it just helps to share common values between documentation and the package builder (*setup* file).

`info.author()`

Returns The main author (last entry of *contributors()*)

`info.contributors()`

Returns A list of all contributors

`info.contributors_str()`

Returns The *contributors()* as comma joined string

`info.email()`

Returns Main *author()* 's mail

`info.pkg_name()`

Returns The package name (on pypi)

`info.release()`

Returns Current release string

Current 0.4

`info.url()`

Returns The repo url (on github)

`info.version()`

Returns Current version string

Current 0.4 (Release: 0.4)

i

`info`, [29](#)

m

`meta`, [11](#)

p

`photon`, [12](#)

s

`settings`, [7](#)

t

`tools.git`, [14](#)

`tools.mail`, [16](#)

`tools.ping`, [17](#)

`tools.signal`, [20](#)

`tools.template`, [20](#)

u

`util.files`, [21](#)

`util.locations`, [23](#)

`util.structures`, [24](#)

`util.system`, [25](#)

Symbols

`_Signal__signal()` (tools.signal.Signal method), 20
`_checkout()` (tools.git.Git method), 14
`_get_branch()` (tools.git.Git method), 14
`_get_remote()` (tools.git.Git method), 14
`_log()` (tools.git.Git method), 14
`_pull()` (tools.git.Git method), 15

A

`alarm` (tools.signal.Signal attribute), 20
`author()` (in module info), 29

B

`backup_location()` (in module util.locations), 23
`branch` (tools.git.Git attribute), 15

C

`change_location()` (in module util.locations), 23
`check_m()` (in module photon), 14
`cleanup` (tools.git.Git attribute), 15
`commit` (tools.git.Git attribute), 15
`contributors()` (in module info), 29
`contributors_str()` (in module info), 29

D

`dict_merge()` (in module util.structures), 24

E

`email()` (in module info), 29
environment variable
 `XDG_CONFIG_HOME`, 23
 `XDG_DATA_HOME`, 23

G

`get` (settings.Settings attribute), 8
`get_hostname()` (in module util.system), 25
`get_locations()` (in module util.locations), 23
`get_timestamp()` (in module util.system), 25
`Git` (class in tools.git), 14
`git_handler()` (photon.Photon method), 12

H

`hup` (tools.signal.Signal attribute), 20

I

`info` (module), 29
`int` (tools.signal.Signal attribute), 20

K

`kill` (tools.signal.Signal attribute), 20

L

`load()` (meta.Meta method), 11
`load()` (settings.Settings method), 8
`local` (tools.git.Git attribute), 15
`log` (meta.Meta attribute), 11
`log` (tools.git.Git attribute), 15

M

`m()` (photon.Photon method), 12
`Mail` (class in tools.mail), 16
`mail_handler()` (photon.Photon method), 13
`make_locations()` (in module util.locations), 23
`Meta` (class in meta), 11
`meta` (module), 11

P

`Photon` (class in photon), 12
`photon` (module), 12
`Ping` (class in tools.ping), 17
`ping_handler()` (photon.Photon method), 13
`pkg_name()` (in module info), 29
`probe` (tools.ping.Ping attribute), 18
`publish` (tools.git.Git attribute), 15

Q

`quit` (tools.signal.Signal attribute), 20

R

`raw` (tools.template.Template attribute), 21

`read_file()` (in module `util.files`), 21
`read_json()` (in module `util.files`), 22
`read_yaml()` (in module `util.files`), 22
`release()` (in module `info`), 29
`remote` (`tools.git.Git` attribute), 15
`remote_url` (`tools.git.Git` attribute), 15

S

`s2m` (`photon.Photon` attribute), 13
`search_location()` (in module `util.locations`), 24
`send` (`tools.mail.Mail` attribute), 16
`Settings` (class in `settings`), 7
`settings` (module), 7
`shell_notify()` (in module `util.system`), 26
`shell_run()` (in module `util.system`), 26
`short_commit` (`tools.git.Git` attribute), 15
`Signal` (class in `tools.signal`), 20
`signal_handler()` (`photon.Photon` method), 13
`stage()` (`meta.Meta` method), 11
`status` (`tools.git.Git` attribute), 15
`status` (`tools.ping.Ping` attribute), 18
`stop` (`tools.signal.Signal` attribute), 20
`sub` (`tools.template.Template` attribute), 21

T

`tag` (`tools.git.Git` attribute), 16
`Template` (class in `tools.template`), 20
`template_handler()` (`photon.Photon` method), 13
`text` (`tools.mail.Mail` attribute), 16
`to_list()` (in module `util.structures`), 24
`tools.git` (module), 14
`tools.mail` (module), 16
`tools.ping` (module), 17
`tools.signal` (module), 20
`tools.template` (module), 20

U

`url()` (in module `info`), 29
`usr1` (`tools.signal.Signal` attribute), 20
`usr2` (`tools.signal.Signal` attribute), 20
`util.files` (module), 21
`util.locations` (module), 23
`util.structures` (module), 24
`util.system` (module), 25

V

`version()` (in module `info`), 29

W

`write()` (`tools.template.Template` method), 21
`write_file()` (in module `util.files`), 22
`write_json()` (in module `util.files`), 22
`write_yaml()` (in module `util.files`), 22

X

`XDG_CONFIG_HOME`, 23
`XDG_DATA_HOME`, 23

Y

`yaml_loc_join()` (in module `util.structures`), 25
`yaml_str_join()` (in module `util.structures`), 25